Developing a 2D game in Java ME - Part 6

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons:
Image capture code doesn't work on Nokia Asha 305.

This article shows how to use the Java Mobile Multimedia API (MMAPI) to add sound to the Java ME game. This is the sixth in a series of articles that covers all the basics of developing an application for mobile devices using Java ME, learning the main base libraries, classes, and methods available in Java ME.

Introduction

24 Aug 2008

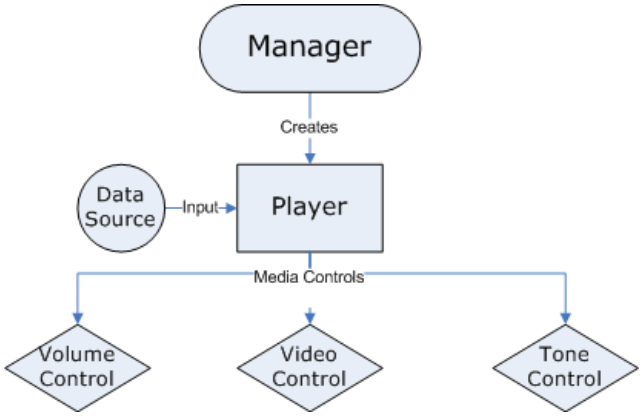
The previous article described how to save game settings, including the Sound On/Off screen, but the game does not have any sound yet. This article describes the Java Mobile Multimedia API (MMAPI) and explains how to add sound to the game.

The MMAPI offers a set of multimedia capabilities for mobile devices, including playback and recording audio and video data from a variety of sources. Of course, not all mobile devices support all the options, but MMAPI is designed to take full advantage of the capabilities that are available and ignore those that it does not support.

MMAPI information

The MMAPI is built on a high-level abstraction of all the multimedia devices. This abstraction is implemented in three classes that form the core of the operations that you do with this API. These classes are the Player and Control interfaces and the Manager class. Another class, the DataSource abstract class, is used to locate resources, but unless you define a new way of reading data you will probably never need to use it directly.

In a nutshell, the Manager class is used to create Player instances for different media by specifying DataSource instances. The Player instances thus created can be configured by using Control instances. For example, almost all Player instances would, in theory, support VolumeControl to control the volume of the Player. Check the following diagram:



The Manager class is basically a factory of players supporting the following creation methods:

- createPlayer(DataSource source): Creates a player based on a DataSource.
- createPlayer(InputStream stream, String type): Creates a player using the input stream as source and assuming that the media type provided. For a list of media types check the IANA Web site.
- createPlayer(String locator): Creates a player using a URL type parameter to identify the source data.

The last method allows you to allocate different types of media, depending on the selected URL protocol. The following types are supported:

- Midi Player - "device://midi": Creates a midi Player.
- Tone Player - "device://tone": Creates a tone Player.
- Capture Audio - "capture://audio": Allows capturing audio from the device microphone.
- Capture Video - "capture://video": Allows capturing video from the device camera.
- Capture Radio - "capture://radio?f=105.1&st=stereo": Allows capturing radio.

(C) Copyright Nokia 2014. All rights reserved.

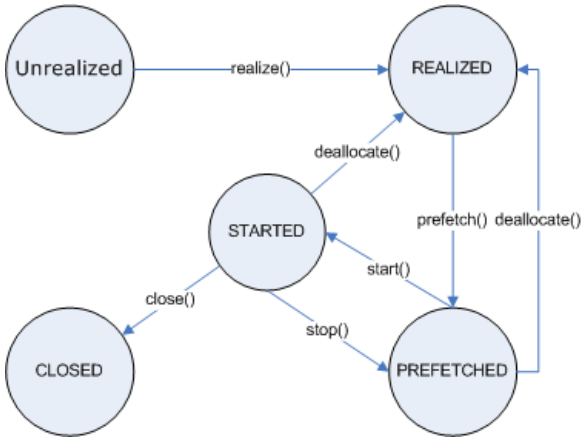
To find out what content types and protocols your device supports, use the following methods of the Manager class:

- getSupportedContentTypes(): Provides a list of available content types for all protocols or for a specific protocol.
- getSupportedProtocols(): Provides a list of available protocols for all content types or for a specific content type.

After you have created a Player, you can start using it by simply calling the `start()` method. When it reaches the end of the media it stops automatically. This is a simplified view of the Player class. Actually the class has five states:

- UNREALIZED: This is the initial state of the Player when obtained from the Manager.
- REALIZED: When `realized()` is called, the Player switches to this state obtaining the information needed to acquire the media resources. Realizing a Player can be a resource and time consuming process. The Player may have to communicate with a server, read a file, or interact with a set of objects.
- PREFETCHED: After a player is realized, it may still need to acquire scarce or exclusive resources, fill buffers with media data, or perform other start-up processing. This is done by calling the `prefetch()` method that switches the player to this state.
- STARTED: When `start()` is called, the Player starts to play the media resource until it reaches the end of the media.
- CLOSED: When `close()` is called, the Player switches to this state, releasing all the resources acquired. It cannot be used again.

The following figure shows the various states and the possible transitions between them:



If your application needs information about the state changes, you need to implement the PlayerListener interface.

Play a sound

In the Arkanoid clone, the idea is to play a sound each time the ball hits a brick or the pad. To do this, create a class called Multimedia with the method `playSound()`:

```
//multimedia libraries
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import javax.microedition.media.MediaException;
public class Multimedia {
  public void playSound(String file, String format) {
    try {
      InputStream is = getClass().getResourceAsStream(file);
      Player p = Manager.createPlayer(is, format);
      p.start();
    } catch (IOException ioe) {
    } catch (MediaException me) {
    }
  }
```

Now just use this method each time a collision between the ball and the other entities is detected. For this purpose, I have created a sound named "click.wav", and it is available in the resource folder.

```
public void updateGameState(){
  ...
```


(C) Copyright Nokia 2014. All rights reserved.

```
  byte colision = ball.colided(pad);
  if (colision != Entity.COLLISION_NONE){
      if (midlet.soundOn){
        midlet.multimedia.playSound("click.wav", "audio/X-wav");
      }
    }
  ...
}
```

If you run the application, you will hear some sound when playing the game.

Capture video

Now that the game has sound, you can capture the player's photo each time he or she achieves the highest score. To do this, you first need to access the video camera and show it to the player. The following method captures the camera's video stream to an Item.

```
  Player p;
  VideoControl vc;
  public Item showVideo(String url){
    Item result = null;
    try {
        p = Manager.createPlayer(url);
        p.realize();
        // Grab the video control .
        vc = (VideoControl)p.getControl("VideoControl");
        if (vc != null) {
            // create the Item with the video image
            result =((Item)vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null));
            // add a label
            result.setLabel("Photo");
        }
        // start capture
        p.start();
    } catch (IOException ioe) {
    } catch (MediaException me) { }
    return result;
  }
```

VideoControl is used to create the Item to be use in the Form.

```
public Displayable initNewHighScore(int score, int pos) {
  ...
  newHighScoreForm.append(multimedia.showVideo("capture://video"));
  ...
}
```

Now use VideoControl to capture an image from the camera:

```
  public Image captureVideo(){
    Image result = null;
    try {
        // grab data
        byte[] imageData = vc.getSnapshot("encoding=png");
```


(C) Copyright Nokia 2014. All rights reserved.

```
        // create image;
        result = Image.createImage(imageData, 0, imageData.length);
    } catch (MediaException me) {
      me.printStackTrace();
    }
    return result;
  }
```

and then call this method when a high score is saved.

```
  // we added an extra field to Score to store the image
  scores[pos].image = multimedia.captureVideo();
```

After this, show the images in the high scores screen.

```
public Displayable initScoreForm() {
...
  if (scores[i].image != null){
    highScoreForm.append(scores[i].image);
  }
...
}
```

Run the application to test the new feature. The next article explains how to use the network capabilities of the phone.

Downloads

- Full Source Code
- Jad File
- Jar File

Go To Part 7

(C) Copyright Nokia 2014. All rights reserved.